

# OCR A Level

Computer  
Science

H446 – Paper 1

3

## Lists and linked lists

Unit 7  
Data structures



PG ONLINE

# Objectives

- Explain how a list may be implemented as a static or dynamic data structure
- Describe the linked list data structure
- Show how to create, traverse, add data to and remove data from a linked list

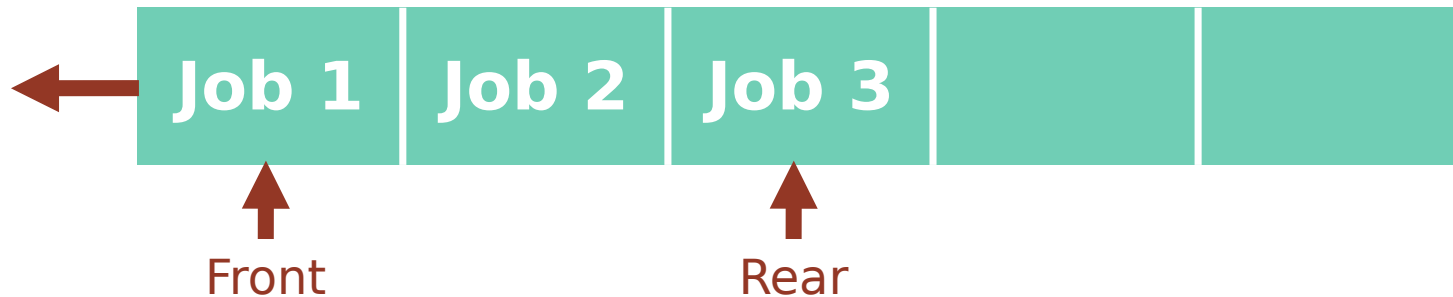


# Abstraction

- All programming languages have data types such as **real**, **integer**, **character** and operations which can be performed on them
- In order to deal with complex problems, we need more complex data types that will lead to more efficient problem-solving methods
- This is the idea behind the creation of **abstract data types**

# Abstraction

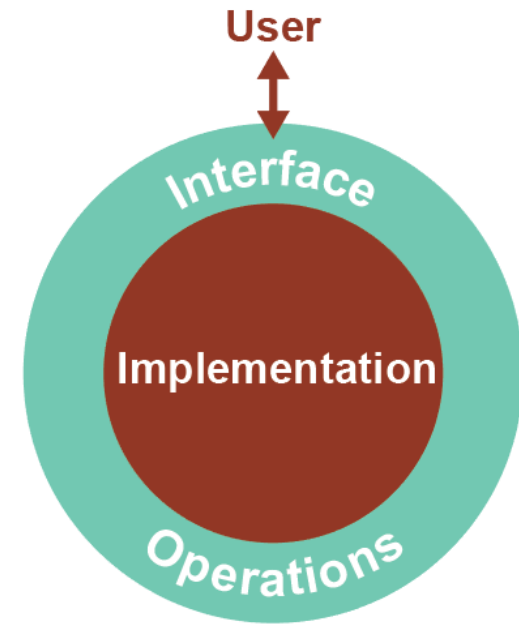
- A **queue** is an Abstract Data Type (ADT)



- The **methods** used to implement the queue (e.g. enqueue, dequeue) can be used without knowledge of how they work
- The **data** (e.g. the pointers) used in implementing a queue are hidden from the user

# List - An ADT

- A **list** is another example of an Abstract Data Type
- Many languages (e.g. Python) have a built-in **list** data type
- An abstract data type (ADT) allows us to view the data and perform operations that are allowed without regard to how they will be implemented



# Dynamic vs static

- In relation to size, what does static mean?
- In relation to size, what does dynamic mean?

# Dynamic vs static

- A static data structure cannot change size after it has been created
- A dynamic data structure can grow or shrink
- Programming languages often have an inbuilt dynamic **list**
- Python, Java, VB.Net, and Delphi all have dynamic list support
  - What controls how many items can be added to a dynamic list?

# Implementation of a dynamic list

- The implementation of an inbuilt **list** ADT is hidden from the user
- A new location is taken from the **'heap'** - memory locations used for dynamic allocation
- When an item is deleted from a list, the memory location is freed up and returned to the **'heap'**
- A system of pointers keeps the list in the order specified by the user



# Applications of lists

- What are some examples of lists in real life and information processing systems?



The image shows an airport arrivals board with a yellow header and black panels. The header has a circular logo with an airplane and the word "Arrivals". The board displays flight information for various destinations, including Dublin, Zurich, Stockholm, Oslo, Berlin, Geneva, Hamburg, Vienna, Frankfurt, Dusseldorf, Belfast City, Shannon, Stavanger, Singapore, Cologne, Edinburgh, Taipei, Zurich, Aberdeen, Bangkok, Dublin, Cairo, Stockholm, Edinburgh, Toronto, Frankfurt, Istanbul, Dublin, Brussels, Athens, Cork, Copenhagen, Geneva, Vienna, Lisbon, Newark, Stockholm, Oslo, Zurich, Warsaw, Washington, and Dublin. The board also shows expected arrival times and flight status (e.g., "Bags delivered", "Arrived", "Expected").

Destination	Time	Status
Dublin	17:50	Bags delivered
Zurich	17:50	Bags delivered
Stockholm	17:50	Bags delivered
Oslo	18:30	Arrived 18:51
Berlin	18:35	Bags delivered
Geneva	18:35	Bags arriving
Hamburg	18:40	Bags arriving
Vienna	18:40	Arrived 18:53
Frankfurt	18:45	Arrived 18:51
Dusseldorf	18:50	Landed 19:01
Belfast City	18:50	Bags delivered
Dublin	18:50	Arrived 19:02
Shannon	18:50	Arrived 19:00
Stavanger	18:50	Arrived 19:01
Singapore	19:00	Expected 19:10
Cologne	19:05	Expected 19:18
Edinburgh	19:05	Expected 19:03
Taipei	19:10	Expected 19:14
via: Bangkok	19:10	Expected 19:27
Zurich	19:25	Expected 19:24
Aberdeen	19:25	Expected 19:29
Bangkok	19:35	Expected 19:24
Dublin	19:50	Expected 19:46
Cairo	19:55	Expected 20:09
Stockholm	20:10	Expected 19:51
Edinburgh	20:20	Expected 19:55
Toronto	20:30	Expected 19:58
Frankfurt	20:45	Expected 19:57
Istanbul	21:00	Expected 21:11
Dublin	21:05	Expected 21:25
Brussels	21:10	Expected 21:05
Athens	21:15	Expected 21:17
Cork	21:15	Expected 21:19
Copenhagen	21:15	Expected 21:21
Geneva	21:20	Expected 21:23
Vienna	21:25	Expected 21:25
Lisbon	21:25	Expected 21:27
Newark	21:25	Expected 21:29
Stockholm	21:35	Expected 21:31
Oslo	21:35	Expected 21:33
Zurich	21:40	Expected 21:35
Warsaw	21:50	Expected 21:37
Washington	22:00	Expected 21:39
Dublin	22:05	Expected 21:41
Frankfurt	22:15	Expected 21:43
Istanbul	22:15	Expected 21:45
Funchal	22:30	Expected 21:47
Lisbon	22:40	Expected 21:49
Munich	22:50	Expected 21:51
Singapore	05:55	Expected 21:53
Chicago	05:55	Expected 21:55
Bangkok	06:20	Expected 21:57
Newark	06:20	Expected 21:59
Johannesburg	06:25	Expected 22:01
Washington	06:50	Expected 22:03
Addis Ababa	06:55	Expected 22:05
Houston	06:55	Expected 22:07

# Applications of lists

- List of students in a class and their marks, component parts of a product, songs, friends, items in a queue, etc.
- “Items in a queue” suggests that we could use the **list** ADT to implement a **queue**, with added conditions specified
  - In general, what operations would it be useful to include in a list?

# Programming operations

- Can you suggest some operations needed to implement a list? Here are two to get you going

List operation	Operation
isEmpty()	Test for empty list
append(item)	Add a new item to the end of a list

# Programming operations

List operation	Operation
isEmpty()	Test for empty list
append(item)	Add a new item to the end of a list
remove(item)	Remove first occurrence of an item from list
count(item)	Return the number of occurrences of item in list
len(item)	Return the number of items in the list
index(item)	Return the position of item
insert(pos,item)	Add a new item at position pos
pop()	Remove and return the last item in the list
pop(pos)	Remove and return the item at position pos



# Worksheet 3

- Complete the 'Random Clothing' **Task 1** on the worksheet



# Sorting a list

- You will cover sorting methods in due course
  - In Python, there is a built-in method for sorting a list, which you can try in interactive mode

```
>>> numbers = [4, 78, 66, 2, 99, 37]
>>> numbers.sort()
>>> numbers
[2, 4, 37, 66, 78, 99]
```

# Implementing a queue as a list

- Using a dynamic data structure such as a list to implement a queue, is there any point in holding items in a circular queue?
- Is it necessary to update the **size** variable as items are added and removed?
- Do we need a variable **maxSize**?
- Do we need a function **isFull**?

# Functions to implement a linear queue as a dynamic list

Write pseudocode to implement the following operations for a queue which can hold a maximum of `maxSize` items:

- `enqueue`
- `dequeue`
- `isEmpty`
- `isFull`



# Pseudocode

```
procedure enqueue(item)
    if q.isFull() then
        print ("queue
full")
    else
        q.append(item)
    endif
endprocedure
```

```
function isFull()
    return (len(q) ==
maxSize)
endfunction
```

```
procedure dequeue(item)
    if q.isEmpty() then
        print ("queue
empty")
    else
        q.pop(0)
    endif
endprocedure
```

```
function isEmpty()
    return (len(q) == 0)
endfunction
```

# Operations on lists

- Merging, sorting, searching and comparing lists are very common operations in computing
  - How could you find how many times numbers in the range 80 -100 occur in a list of unsorted integers?
  - How could you remove all these numbers from the list?

# Worksheet 3

- Try these operations in **Task 2** on the worksheet



# Linked lists

- A dynamic abstract data structure which can be implemented as an array and pointers
- Composed of 'nodes'
- Each node is composed of two parts
  - The data (which may be a complex data structure)
  - A pointer (the index) of the next node
- A **start** pointer identifies the first node in the list
- A **nextfree** pointer shows the index of the next free space in the array



# Array implementation

- The empty array is initialised as a linked list of free spaces
- **start** will point to the first element in the list

Index	Data	Pointer
0		1
1		2
2		3
3		4
4		null

start

null

nextfree

0

# Adding elements to the list

- We will add the names Nancy, Ava, Dave, Peter to the list
- Start with Nancy

Index	Data	Pointer
0	Nancy	null
1		2
2		3
3		4
4		null

start

0

nextfree

1

# Adding elements to the list

- We will add the names Nancy, Ava, Dave, Peter to the list
- Now add Ava

Index	Data	Pointer
0	Nancy	null
1	Ava	0
2		3
3		4
4		null

start

1

nextfree

2

# Adding elements to the list

- We will add the names Nancy, Ava, Dave, Peter to the list
- Now add Dave

Index	Data	Pointer
0	Nancy	null
1	Ava	2
2	Dave	0
3		4
4		null

start

1

nextfree

3

# Adding elements to the list

- We will add the names Nancy, Ava, Dave, Peter to the list
- Now add Peter. What will be the state of the array and the pointers?

Index	Data	Pointer
0	Nancy	null
1	Ava	2
2	Dave	0
3		4
4		null

start

1

nextfree

3



# Adding elements to the list

- We will add the names Nancy, Ava, Dave, Peter to the list
- Now Peter has been added

Index	Data	Pointer
0	Nancy	3
1	Ava	2
2	Dave	0
3	Peter	null
4		null

start

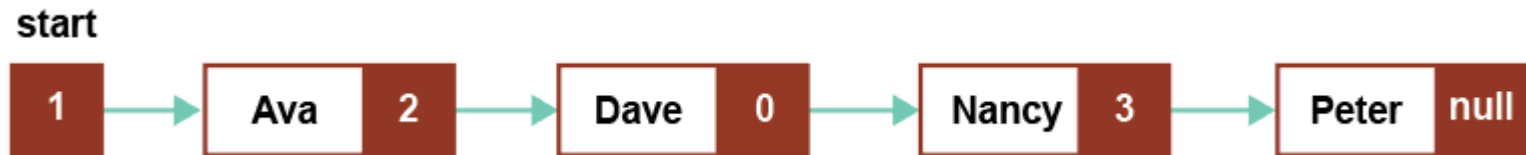
1

nextfree

4

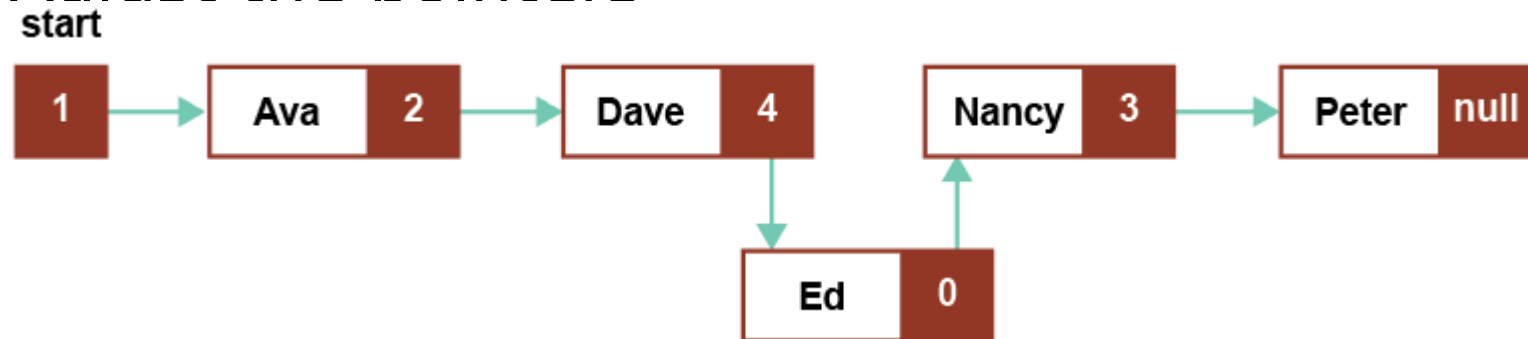
# Linked List - diagram

- **start** points to the head of the list
- Each pointer field holds the index of the next node
- Last node has a **null** pointer



# Adding a new node

- Put the data in the node pointed to by nextfree
- Follow the pointers to find where the new node needs to be linked in
- Adjust the pointers



# Deleting a node

- To delete a node, we just need to adjust the pointers
- The deleted node can be linked back in to the list of free nodes by adjusting the pointer in **nextfree** and the pointer in the deleted node



# Peeking ahead

- We can examine the data and the pointer in the current node **p** and the next one.

- Suppose  $p = 1$
- $List[p].Data = Ava$
- $List[p].Pointer = 2$
- $next = List[p].pointer$
- What is  $List[next].data$ ?
- What is  $List[next].pointer$ ?

Index	Data	Pointer
0	Nancy	3
1	Ava	2
2	Dave	0
3	Peter	null
4		null

start

1

nextfree

4





# Peeking ahead

- We can examine the data and the pointer in the current node **p** and the next one.

- Suppose  $p = 1$
- $List[p].Data = Ava$
- $List[p].Pointer = 2$
- $next = List[p].pointer$
- $List[next].data = Dave$
- $List[next].pointer = 0$

- This technique is used in processing a linked list

Index	Data	Pointer	start
0	Nancy	3	1
1	Ava	2	nextfree
2	Dave	0	4
3	Mike	null	
4		null	

# Worksheet 3 - Operations

- Complete **Task 3** on the worksheet to develop the algorithms for:
  - Inserting a node
  - Deleting a node

# Plenary

- A dynamic data structure such as list is useful for implementing other ADTs such as queues, stacks and trees
  - What is the difference between a static and dynamic data structure?
  - What operations can be performed on a dynamic list?
- You should practise writing and tracing through algorithms for processing a linked list implemented as an array of records

## **Copyright**

© 2016 PG Online Limited

The contents of this unit are protected by copyright.

This unit and all the worksheets, PowerPoint presentations, teaching guides and other associated files distributed with it are supplied to you by PG Online Limited under licence and may be used and copied by you only in accordance with the terms of the licence. Except as expressly permitted by the licence, no part of the materials distributed with this unit may be used, reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic or otherwise, without the prior written permission of PG Online Limited.

## **Licence agreement**

This is a legal agreement between you, the end user, and PG Online Limited. This unit and all the worksheets, PowerPoint presentations, teaching guides and other associated files distributed with it is licensed, not sold, to you by PG Online Limited for use under the terms of the licence.

The materials distributed with this unit may be freely copied and used by members of a single institution on a single site only. You are not permitted to share in any way any of the materials or part of the materials with any third party, including users on another site or individuals who are members of a separate institution. You acknowledge that the materials must remain with you, the licencing institution, and no part of the materials may be transferred to another institution. You also agree not to procure, authorise, encourage, facilitate or enable any third party to reproduce these materials in whole or in part without the prior permission of PG Online Limited.

